Database programming of highly intelligent robots serving elderly semi-paralyzed patients 2025v1.5

Database Programming of High Intelligent Robots Serving Elderly Semi-Paralyzed Patients 2025V1.5 ●●● A database of program codes of high intelligent robots serving elderly semi-paralyzed patients. The code programs include various detailed data of multi-modal and multi-modal scenes such as robot serving elderly semi-paralyzed patients, feeding and medicine, washing face, washing dishes and mopping the floor, cooking and cooking, cutting vegetables and washing vegetables. Based on the development requirements of multi-modal robot nursing system, combined with the technical practice in the field of medical robots and intelligent old-age care, the following is the core implementation scheme of program framework and database design. Focus on solving the life care scenes of the semi-paralyzed elderly:-# # * * 1. Database design (MySQL example) * **```sql-- patient information table create table patients (patient _ id int primary key auto _ increment, Name varchar (50) not null, age int, paralysis _ levelenum ('mild',' moderate',' severe') COMMENT' paralysis level', medical _ historytext, created _ at timestamp default current _ timestamp); -action instruction library create table actions (action _ id int primary key auto _ increment, name varchar (50) unique not null, -such as "feeding medicine", "turning over" safety_threshold FLOAT COMMENT' strength/angle safety threshold', default_duration INT COMMENT' default execution time (seconds)'); -multimodal scene data table create table multimodal _ scenes (scene _ id int primary key, action _ id int, sensor _ data JSON comment' {"force _ sensor" ":0.5, "vision":"dish_position"}', environment_factors JSON COMMENT '{"light":300,"obstacles":["chair","table"]}', FOREIGN KEY (action_id) REFERENCES Actions(action_id)); -personalized care plan create tablecare _ plans (plan _ id int primary key, patient _ id int, schedule JSON comment' {"time": "08: 00", "action": "medication", "medicine_type": "capsule"}', adaptive _ params JSON comment' {"head _ elevation _ angle": 30, "spoon _ speed": 0.2}', foreign key (patient _ id) references patients (patient _ id)); ```-# # * * Second, the core program module (Python pseudocode) * * # # 1. * * Action control engine * * ````` Python class nursing robot: def _ _ init _ _ (self, Patient _ id): self. patient = load _ patient _ data (patient _ id) # Load patient data from database self. sensors = multimodalsensorsuite () # Multi-modal sensor group def execute_action(self, Action _ name): action = db. query _ action (action _ name) scene _ data = self. sensors. get _ real _ time _ data () # Get real-time environmental data # Security check (based on [1] (_blog.csdn.net/hongfenge 123/article/details/144814166) if not self._safety_check(action, Scene _ data): raise safety violation ("force control or environmental abnormality") # Call hardware execution (example: drug administration) if action _ name = = "feed _ medicine": self.arm.set _ force _ limit (action.safety _ threshold). Self.vision.locate_mouth() # Visually locate mouth self. arm. move _ monument (calc _ monument (scene _ data)) self.dispenser.release_medicine() def _safety_check(self, action, Sensor_data): ""according to [9] (_news.cn/politics/202506_ ec04001e7b428bc147b6aeaca81b/c.html). Force feedback and visual fusion technology based on ""return (sensor _ data ["force"] < action. safety _ threshold and sensor _ data ["occlusion _ distance"] > 10.0) ``` # # # 2. * * Environmental interaction module (kitchen scene example) * * ```` Python class kitchen task: defcook _ meal (self, Menu): ingredients = self. _ prepare _ ingredients (menu) # linked vegetable cutting/ The vegetable washing robot forstep in menu.steps: ifstep = = "stir _ fry": self. _ adjust _ stove _ temperature (step.temp) # Safety monitoring based on thermal imaging sensor self._monitor_smoke() # (refer to

□□□□ FINGER = 3 # □□□□□□# □□□□□□□ class RobotState(Enum): IDLE = 0
COOKING = 1 CLEANING = 2 HELPING = 3 COMMUNICATING = 4 MOVING = 5
MEDICATING = 6# □□□□□□□□□□ class FingerControl: def __init__(self):
self.finger_positions = [0.0] * 5 # □□□□□□□0.0-1.0 □□ self.gripping_force = 0.0 # □
□□□□ def set_finger_position(self, finger_idx: int, position: float) -> None: """□□□□
□□□□□""" if 0 <= finger_idx < 5 and 0.0 <= position <= 1.0:
self.finger_positions[finger_idx] = position logger.info(f"Finger {finger_idx} set to
{position:.2f}") else: logger.error("Invalid finger index or position") def
grip_object(self, object_weight: float) -> None: """□□□□□□□□□□□"""
self.gripping_force = min(1.0, object_weight * 0.3) # □□□□□□□□
logger.info(f"Gripping force set to {self.gripping_force:.2f} for weight
{object_weight}kg") def pick_up_object(self, object_type: str, position:
Tuple[float, float, float]) -> bool: """□□□□□□□□□□□□□□□□""" logger.info(f"Picking up
{object_type} at position {position}") # 1. □□□□ for i in range(5):
self.set_finger_position(i, 0.8) time.sleep(0.1) # 2. □□□□□□□ # □□□□□□□□□□□□□□ #
3. □□□□□□□□□□ for i in range(5): self.set_finger_position(i, 0.2 + (0.1 * i)) # □□□□□
□□□□ time.sleep(0.05) self.grip_object(0.5 if object_type == "cup" else 1.0) # □□□
□ 0.5kg return True# □□□□□□□ class JointControl: def __init__(self):
self.joint_angles = {joint: 0.0 for joint in JointType} self.motion_speed = 1.0 # □□
□□□□ def set_joint_angle(self, joint: JointType, angle: float, speed: float = None) -
> None: """□□□□□□□□""" if speed is None: speed = self.motion_speed
self.joint_angles[joint] = angle logger.info(f"Joint {joint.name} set to {angle:.2f}
degrees at speed {speed:.2f}") def move_joints(self, joint_angles: Dict[JointType,
float], speed: float = None) -> None: """□□□□□□□□""" if speed is None: speed =
self.motion_speed # □□□□□□□□□□□□□□□□ max_change = 0 current_angles =
self.joint_angles.copy() for joint, angle in joint_angles.items(): max_change =
max(max_change, abs(angle - current_angles[joint])) # □□□□□□ steps =
int(max_change * 10 / speed) # □□ 10 □□□□□□□□ for step in range(steps + 1): t =
step / steps for joint, target_angle in joint_angles.items(): self.joint_angles[joint]
= current_angles[joint] + t * (target_angle - current_angles[joint])
time.sleep(0.05) # □□□□ def perform_motion(self, motion_type: MotionType,
params: Dict) -> None: """□□□□□□""" if motion_type == MotionType.FINGER: # □□
□□□ FingerControl □□ pass elif motion_type == MotionType.LINEAR: # □□□□□□□□□
distance = params.get('distance', 0.5) direction = params.get('direction', [1, 0,
0]) self._walk_linear(distance, direction) # □□□□□□□□... def _walk_linear(self,
distance: float, direction: List[float]) -> None: """□□□□□□""" logger.info(f"Walking
{distance}m in direction {direction}") steps = int(distance / 0.2) # □□□ 0.2 □ for
step in range(steps): # □□□□□□□ if step % 2 == 0:
self.set_joint_angle(JointType.ANKLES, 15.0) # □□□□ else:
self.set_joint_angle(JointType.ANKLES, -15.0) # □□□□ time.sleep(0.5) # □□□□ # □□
□□□□ self.set_joint_angle(JointType.ANKLES, 0.0)# □□□□□□□ class NLPInteraction:
def __init__(self): self.conversation_history = [] self.emotion_recognition =
{ 'happy': 0.0, 'sad': 0.0, 'neutral': 1.0, 'frustrated': 0.0 } def
start_conversation(self, topic: str) -> str: """□□□□"""
self.conversation_history.append(f"Robot: Hello! Would you like to talk about
{topic}?") logger.info(f"Started conversation on topic: {topic}") return "Hello!
Would you like to talk about " + topic + "?" def respond_to_input(self, user_input:
str) -> str: """□□□□□□""" # □□□□□□□□□□□□□□□□□ NLP □□
self.conversation_history.append(f"User: {user_input}") # □□□□□□□ if "good" in
user_input or "happy" in user_input: self.emotion_recognition['happy'] += 0.1
self.emotion_recognition['neutral'] -= 0.1 elif "bad" in user_input or "sad" in
user_input: self.emotion_recognition['sad'] += 0.1
self.emotion_recognition['neutral'] -= 0.1 # □□□□ if "music" in user_input:
response = "Yes, music is wonderful. Would you like to listen to a particular

song?" elif "newspaper" in user_input: response = "The nurse will bring the newspaper soon. Would you like me to read it to you?" elif "walk" in user_input: response = "That's a great idea! Let me help you get ready for a walk." else: response = "That's interesting. Can you tell me more?" self.conversation_history.append(f"Robot: {response}") logger.info(f"Responded: {response}") return response def play_music(self, genre: str = "classical") -> None: """□□□□""" logger.info(f"Playing {genre} music") # □□□□□□□□□□□ # □□□□□□ print(f"[Music playing: {genre} melody □□...]") time.sleep(2) # □□□□ def read_newspaper(self, article: str) -> None: """□□□""" logger.info(f"Reading newspaper article: {article[:20]}...") # □□□□□□□□□□ # □□□□□□ print(f"[Reading newspaper: {article}]")# □□□□□□□□□ class TaskScheduler: def __init__(self): self.current_task = None self.task_queue = [] self.robot_state = RobotState.IDLE self.joint_control = JointControl() self.finger_control = FingerControl() self.nlp = NLPInteraction() def add_task(self, task: str, params: Dict = None) -> None: """□□□□□□□""" if params is None: params = {} self.task_queue.append((task, params)) logger.info(f"Task added: {task}, params: {params}") self._process_tasks() def _process_tasks(self) -> None: """□□□□□□""" if self.current_task is None and self.task_queue: self.current_task = self.task_queue.pop(0) self._execute_task(*self.current_task) def _execute_task(self, task: str, params: Dict) -> None: """□□□□□□""" task_mapping = { "prepare_meal": self._prepare_meal, "do_laundry": self._do_laundry, "clean_floor": self._clean_floor, "feed_meal": self._feed_meal, "give_medicine": self._give_medicine, "help_stand_up": self._help_stand_up, "help_walk": self._help_walk, "help_dress": self._help_dress, "help_wash": self._help_wash, "have_conversation": self._have_conversation, "go_for_walk": self._go_for_walk, "sit_on_chair": self._sit_on_chair, "listen_to_music": self._listen_to_music, "read_newspaper": self._read_newspaper } if task in task_mapping: # □□□□□□□ state_mapping = { "prepare_meal": RobotState.COOKING, "do_laundry": RobotState.CLEANING, "clean_floor": RobotState.CLEANING, "feed_meal": RobotState.HELPING, "give_medicine": RobotState.MEDICATING, "help_stand_up": RobotState.HELPING, "help_walk": RobotState.MOVING, "help_dress": RobotState.HELPING, "help_wash": RobotState.HELPING, "have_conversation": RobotState.COMMUNICATING, "go_for_walk": RobotState.MOVING, "sit_on_chair": RobotState.MOVING, "listen_to_music": RobotState.COMMUNICATING, "read_newspaper": RobotState.COMMUNICATING } self.robot_state = state_mapping.get(task, RobotState.IDLE) logger.info(f"Executing task: {task}, state: {self.robot_state.name}") # □□□□ task_mapping[task](params) # □□□□ self.current_task = None self.robot_state = RobotState.IDLE logger.info(f"Task completed: {task}") self._process_tasks() else: logger.error(f"Unknown task: {task}") def _prepare_meal(self, params: Dict) -> None: """□□ meals□□□□□□□□□□□□""" logger.info("Preparing meal...") # □□□□ print("[Robot: Washing vegetables...]") self.joint_control.set_joint_angle(JointType.ELBOWS, 90.0) self.joint_control.set_joint_angle(JointType.WRISTS, -15.0) for i in range(3): self.finger_control.set_finger_position(0, 0.5) # □□ self.finger_control.set_finger_position(1, 0.5) # □□ time.sleep(0.5) self.finger_control.set_finger_position(0, 0.8) self.finger_control.set_finger_position(1, 0.8) time.sleep(0.5) # □□□□ print("[Robot: Chopping vegetables...]") self.joint_control.move_joints({ JointType.SHOULDERS: 30.0, JointType.ELBOWS: 120.0, JointType.WRISTS: 0.0 }) self.finger_control.grip_object(1.2) # □□□□□□ 1.2kg for i in range(5): self.joint_control.set_joint_angle(JointType.ELBOWS, 90.0) # □□ time.sleep(0.3) self.joint_control.set_joint_angle(JointType.ELBOWS, 120.0) # □□ time.sleep(0.2) # □□□□ print("[Robot: Stir-frying...]")

self.joint_control.move_joints({ JointType.SHOULDERS: 45.0, JointType.ELBOWS: 110.0, JointType.WRISTS: 15.0 }) for i in range(10): # □□□□□□□□ self.joint_control.set_joint_angle(JointType.WRISTS, 15.0 + 30.0 * np.sin(i * 0.628)) time.sleep(0.4) def _do_laundry(self, params: Dict) -> None: """□□□""" logger.info("Doing laundry...") print("[Robot: Loading washing machine...]") self.joint_control.move_joints({ JointType.HIPS: -15.0, # □□ JointType.ELBOWS: 90.0, JointType.WRISTS: 0.0 }) self.finger_control.pick_up_object("clothes", (0.5, 0.3, 0.2)) # □□□□□□ # □□□□□□□□□ time.sleep(2) self.joint_control.set_joint_angle(JointType.HIPS, 0.0) # □□□ print("[Robot: Starting washing machine...]") # □□□□□□□□□ def _clean_floor(self, params: Dict) -> None: """□□""" logger.info("Cleaning floor...") print("[Robot: Mopping the floor...]") self.joint_control.move_joints({ JointType.HIPS: -20.0, # □□□□ JointType.ELBOWS: 100.0, JointType.WRISTS: 0.0 }) # □□□□□□□□ for i in range(8): direction = 1 if i % 2 == 0 else -1 self.joint_control.set_joint_angle(JointType.SHOULDERS, 30.0 * direction) time.sleep(0.6) self.joint_control.set_joint_angle(JointType.HIPS, 0.0) def _feed_meal(self, params: Dict) -> None: """□□""" logger.info("Feeding meal...") print("[Robot: Feeding the elderly...]") # □□□□□□□□□□ self.finger_control.set_finger_position(0, 0.4) # □□ self.finger_control.set_finger_position(1, 0.3) # □□ self.finger_control.set_finger_position(2, 0.3) # □□ self.joint_control.move_joints({ JointType.SHOULDERS: 40.0, JointType.ELBOWS: 80.0, JointType.WRISTS: -10.0 }) # □□□□□□ for i in range(5): # □□□□ self.finger_control.set_finger_position(1, 0.5) # □□□□ time.sleep(0.5) # □□□□□ self.joint_control.set_joint_angle(JointType.ELBOWS, 60.0) time.sleep(0.5) # □□□□ self.joint_control.set_joint_angle(JointType.WRISTS, 10.0) time.sleep(0.3) # □□ self.joint_control.set_joint_angle(JointType.ELBOWS, 80.0) self.joint_control.set_joint_angle(JointType.WRISTS, -10.0) time.sleep(0.5) def _give_medicine(self, params: Dict) -> None: """□□""" logger.info("Giving medicine...") pill_count = params.get('pill_count', 1) print(f"[Robot: Giving {pill_count} pills...]") # □□□□□□□□ self.finger_control.set_finger_position(0, 0.2) # □□□□ self.finger_control.set_finger_position(1, 0.2) # □□□□ self.joint_control.move_joints({ JointType.ELBOWS: 90.0, JointType.WRISTS: 0.0 }) self.finger_control.pick_up_object("pill", (0.3, 0.2, 0.1)) # □□□□□□ # □□□□□ self.joint_control.set_joint_angle(JointType.ELBOWS, 70.0) time.sleep(0.5) # □□□□ self.finger_control.set_finger_position(0, 0.8) self.finger_control.set_finger_position(1, 0.8) time.sleep(0.3) # □□ self.joint_control.set_joint_angle(JointType.ELBOWS, 90.0) def _help_stand_up(self, params: Dict) -> None: """□□□□□□""" logger.info("Helping stand up...") print("[Robot: Assisting to stand up...]") # □□□□□□□□□□□□ self.joint_control.perform_motion(MotionType.LINEAR, {'distance': 0.5}) # □□□□ self.joint_control.move_joints({ JointType.SHOULDERS: 30.0, JointType.ELBOWS: 160.0, JointType.WRISTS: 0.0 }) # □□□□□□□□ for i in range(5): self.finger_control.set_finger_position(i, 0.6 - 0.1 * i) # □□□□□□□□□□ time.sleep(0.1) # □□□□□□□□□□ # □□□□□□□□□□□□□□□□□□□□ print("[Robot: Applying gentle upward force to assist standing...]") time.sleep(2) # □□□□□□□□ self.joint_control.set_joint_angle(JointType.HIPS, -5.0) # □□□□□□ def _help_walk(self, params: Dict) -> None: """□□□□□□""" logger.info("Helping walk...") distance = params.get('distance', 1.0) print(f"[Robot: Assisting to walk {distance} meters...]") # □□□□□□ self.joint_control.move_joints({ JointType.SHOULDERS: 25.0, JointType.ELBOWS: 150.0, JointType.WRISTS: 5.0 }) # □□□□ for step in range(int(distance / 0.2)): # □□□□□□□□□□ self.joint_control._walk_linear(0.2, [1, 0, 0]) time.sleep(1.0) # □□□□□ def _help_dress(self, params: Dict) -> None: """□□□□□□""" logger.info("Helping

dress...") clothing_type = params.get('clothing_type', "shirt") print(f"[Robot: Helping put on {clothing_type}...]") # □□□□□ if clothing_type == "shirt": # □□□□ self.finger_control.pick_up_object("shirt", (0.4, 0.3, 0.2)) # □□□□ self.joint_control.move_joints({ JointType.SHOULDERS: 60.0, JointType.ELBOWS: 140.0 }) self.finger_control.set_finger_position(0, 0.8) self.finger_control.set_finger_position(1, 0.8) time.sleep(0.5) # □□□□□ print("[Robot: Guiding arm into sleeve...]") self.joint_control.set_joint_angle(JointType.ELBOWS, 120.0) time.sleep(1.0) # □□□□ self.joint_control.set_joint_angle(JointType.WRISTS, -10.0) time.sleep(0.5) def _help_wash(self, params: Dict) -> None: """□□□□□□□□""" logger.info("Helping wash...") print("[Robot: Helping wash face...]") # □□□□ self.finger_control.pick_up_object("towel", (0.3, 0.4, 0.1)) # □□□□ # □□□□ self.finger_control.set_finger_position(0, 0.7) self.finger_control.set_finger_position(1, 0.7) time.sleep(0.5) # □□□□ self.joint_control.move_joints({ JointType.SHOULDERS: 35.0, JointType.ELBOWS: 80.0, JointType.WRISTS: 0.0 }) for i in range(3): # □□□□ self.joint_control.set_joint_angle(JointType.SHOULDERS, 35.0 + 20.0 * (-1) ** i) time.sleep(0.8) # □□□□ self.joint_control.set_joint_angle(JointType.ELBOWS, 120.0) time.sleep(0.5) def _have_conversation(self, params: Dict) -> None: """□□□□□""" topic = params.get('topic', "daily life") logger.info(f"Having conversation on topic: {topic}") print(f"[Robot: Starting conversation about {topic}...]") response = self.nlp.start_conversation(topic) print(f"Robot: {response}") # □□□□□□ for i in range(3): user_response = f"User: That's interesting, tell me more about {topic}." print(user_response) response = self.nlp.respond_to_input(user_response) print(f"Robot: {response}") time.sleep(1.5) def _go_for_walk(self, params: Dict) -> None: """□□□□□□□□""" logger.info("Going for a walk...") print("[Robot: Helping go for a walk in the garden...]") # □□□□ self._help_stand_up({}) # □□□□ self._help_walk({'distance': 2.0}) # □□□□□□□□□□□□□□ self.joint_control.move_joints({ JointType.ELBOWS: 90.0, JointType.WRISTS: 0.0 }) self.finger_control.set_finger_position(2, 0.5) # □□□□ self.finger_control.set_finger_position(3, 0.5) # □□□□□ time.sleep(0.5) self.joint_control.set_joint_angle(JointType.WRISTS, 30.0) # □□□□□ time.sleep(0.5) self.joint_control.set_joint_angle(JointType.SHOULDERS, 45.0) # □□□ time.sleep(1.0) # □□□□ self._help_walk({'distance': 5.0}) def _sit_on_chair(self, params: Dict) -> None: """□□□□□□□□□□""" logger.info("Sitting on chair...") print("[Robot: Assisting to sit on garden chair...]") # □□□□ self.joint_control.perform_motion(MotionType.LINEAR, {'distance': 0.5}) # □□□□ self.joint_control.perform_motion(MotionType.ROTATIONAL, {'angle': 90.0}) # □□□□ print("[Robot: Guiding to sit down gently...]") for i in range(3): self.joint_control.set_joint_angle(JointType.HIPS, -5.0 * i) # □□□□ time.sleep(0.5) # □□□□ self.joint_control.set_joint_angle(JointType.TORSO, 10.0) # □□□□ def _listen_to_music(self, params: Dict) -> None: """□□□□""" genre = params.get('genre', "classical") logger.info(f"Listening to {genre} music...") print("[Robot: Playing beautiful music...]") self.nlp.play_music(genre) # □□□□□□□□□□□□□□ for i in range(5): self.joint_control.set_joint_angle(JointType.HEAD, 10.0 * np.sin(i * 0.628)) time.sleep(1.0) def _read_newspaper(self, params: Dict) -> None: """□□□""" article = params.get('article', "Today's headlines") logger.info(f"Reading newspaper: {article[:20]}...") print("[Robot: Reading the newspaper aloud...]") self.nlp.read_newspaper(article) # □□□□□□ self.joint_control.move_joints({ JointType.ELBOWS: 90.0, JointType.WRISTS: -15.0 }) self.finger_control.set_finger_position(1, 0.3) # □□□□ time.sleep(0.5) self.joint_control.set_joint_angle(JointType.WRISTS, 15.0) # □□ time.sleep(0.5)# □□□□□□□□□□□ class ElderCareRobotSystem: def __init__(self): self.task_scheduler = TaskScheduler() self.is_running = False def start_system(self) -> None: """□□□□□